

AD-A161 984

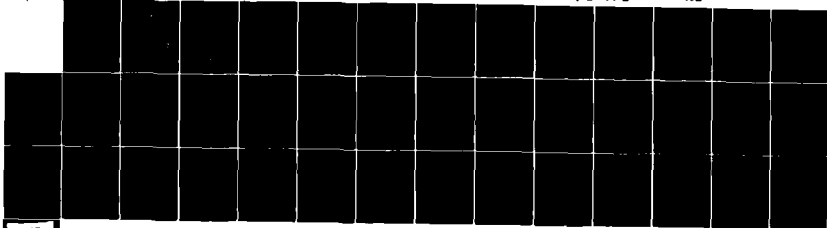
A REAL-TIME DATA ACQUISITION SYSTEM FOR A LOW SPEED
WIND TUNNEL(U) AERONAUTICAL RESEARCH LABS MELBOURNE
(AUSTRALIA) B D FAIRLIE FEB 66 AR-AERO-R-163

1/1

UNCLASSIFIED

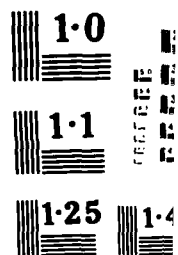
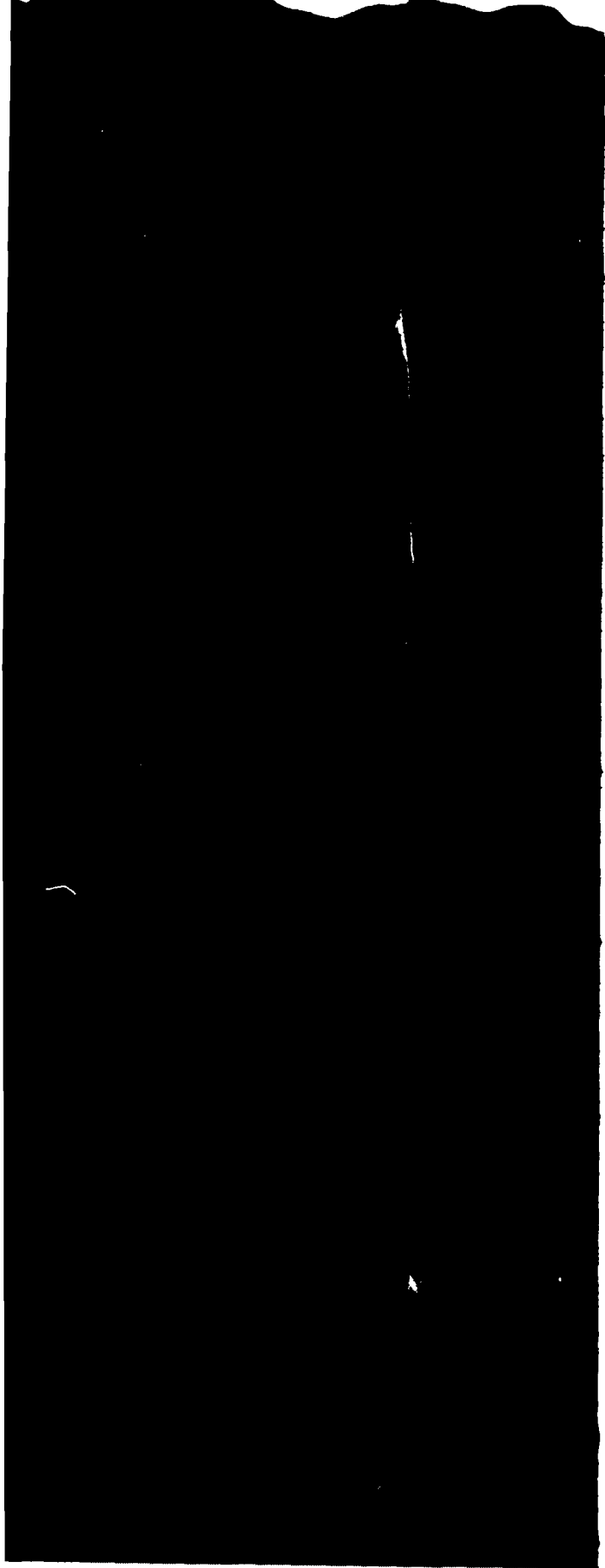
F/O 9/2

NL



END
ONE
FILMED
-86
DTIC

NCLASS



AR-AERO-R-163

AR-003-997

12



AD-A161 984

DEPARTMENT OF DEFENCE
DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
AERONAUTICAL RESEARCH LABORATORIES
MELBOURNE, VICTORIA

AERODYNAMICS REPORT 163

**A REAL-TIME DATA ACQUISITION
SYSTEM FOR A
LOW SPEED WIND TUNNEL**

DTIC
SELECTED
DEC 05 1985
S D

by

B. D. FAIRLIE

THE UNITED STATES NATIONAL
TECHNICAL INFORMATION SERVICE
IS AUTHORIZED TO
REPRODUCE AND SELL THIS REPORT

Approved for Public Release

DTIC FILE COPY

© COMMONWEALTH OF AUSTRALIA 1985

COPY No

FEBRUARY 1985

85 12 3 067

AR-003-997

DEPARTMENT OF DEFENCE
DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
AERONAUTICAL RESEARCH LABORATORIES

AERODYNAMICS REPORT 163

**A REAL-TIME DATA ACQUISITION
SYSTEM FOR A
LOW SPEED WIND TUNNEL**

by

B. D. FAIRLIE

SUMMARY

A mini-computer based real-time data acquisition system designed for use in the Aeronautical Research Laboratories low-speed wind tunnel is presented. The report provides an overview of the logical arrangement of the software components of the system and describes their interaction with the mini-computer operating system, data structures, and system hardware.

© COMMONWEALTH OF AUSTRALIA 1985



POSTAL ADDRESS: Director, Aeronautical Research Laboratories,
Box 4331, P.O., Melbourne, Victoria, 3001, Australia

CONTENTS

	Page No.
1. INTRODUCTION	1
2. OVERALL DESIGN PHILOSOPHY	1
3. HARDWARE	2
4. SERVICES PROVIDED BY THE OPERATING SYSTEM AND EXECUTIVE	4
5. DATA STRUCTURES	6
6. SOFTWARE	7
7. FUTURE DEVELOPMENT	15
8. CONCLUSIONS	17

REFERENCES

FIGURES

APPENDIX A—The Digital Data Bus Input/Output Task [DIGIO]

APPENDIX B—Format of Configuration Files for Force Measuring Tasks

DISTRIBUTION

DOCUMENT CONTROL DATA



Accession For	
NTIS CRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

1. INTRODUCTION

The low-speed wind tunnel at the Aeronautical Research Laboratories is of the closed-jet single return type with a test section of irregular octagonal shape 2.74 metres wide by 2.13 metres high. The tunnel is driven by electric motors with a total output of 0.75 Mw producing wind speeds of up to 100 ms^{-1} . The tunnel was designed in 1939 and construction completed in 1941 since when the tunnel has been in almost continuous service.

Typical of tunnels built in this period, data acquisition and reduction during the early life of the ARL tunnel was almost entirely manual. Through the years these processes have been gradually automated. For example, the installation of a "Flexowriter" in the early 1960's allowed analysis of the data to be carried out on external computers in an off-line "batch" type of operation. However, it was not until the early 1970's that the first true on-line data acquisition system was installed. This system¹ collected input from a wide range of manually selected data sources in a prearranged sequence, and output them (in serial form) to some storage device. Initially, a Teletypewriter with paper tape punch was used as the output device, the paper tape being input to the central site computer at a later time. Eventually, the output was connected directly to the central site computer, allowing true real-time data processing. In its final form, the system included a display screen, with processed data being returned from the central site machine and displayed in near real-time.

By 1980 however, the central site computer had become overloaded, and the time taken for processing wind tunnel data had become intolerably large. The restrictions involved with using the central site processor together with the inflexibility of the overall system was also seriously inhibiting the development of modern testing techniques. It was therefore decided to install a dedicated mini-computer to be used solely for wind tunnel data acquisition and processing, and tunnel control. This mini-computer, a Digital Equipment Corporation PDP-11/44, was installed in July 1982, and following a short period of operation in parallel with the existing system, took over all data acquisition and data reduction for the low-speed wind tunnel.

This report describes the development of the PDP-11/44-based system. The emphasis is on the development of software, although a description of the hardware is included where necessary for a complete understanding of the software design. Consideration is given to the general design principles which have been used in developing the software, and to the design of an efficient data structure. The majority of the software described is already in use, and has been operating satisfactorily in many test programmes. The remaining few facilities which have not as yet been fully implemented, are all under active development. The report concludes with suggestions concerning the directions in which the system should be developed in the future.

2. OVERALL DESIGN PHILOSOPHY

Before describing the development of the system in detail, it will be useful to set out the general philosophy which has been used while designing the system.

Of all the requirements of a well-designed data acquisition system there are two which are of overriding importance:

- (i) the system must be user-friendly and
- (ii) it must be conceptually simple and easy to maintain.

Considering the widely varying levels of skill and aptitude likely to be displayed by potential tunnel operators, the need for user-friendliness is obvious. Any action on the part of a tunnel operator, no matter how unrealistic, should produce a response from the system which allows the test to continue in an orderly manner. Hence in all software, every operator input is checked for validity, and extensive use is made of default replies and error recovery routines. Although it is not possible at present (due to hardware limitations), the long-term aim is that all communication between the tunnel operator and the data system should take place via a single display terminal and keyboard. All commands to the system and all information requiring the operators attention should be available at this single operators terminal. To further centralize system operation, new instrumentation will not include the traditional local display facility. Rather, the outputs of those instruments critical to a particular phase of a test will be displayed on the operator's terminal (see Section 7).

The need for conceptual simplicity and ease of maintenance is also obvious. For various reasons it is very likely that a time will be reached when the original system designers are no longer directly associated with it. The logic of the system must therefore be clear enough to allow their successors to continue to develop and maintain it. Such logical clarity may only be obtained by making use of good modern design methods, including structured programming, modularity, and good programming style. Structured programming is a simple approach to program design which produces programs which are easy to understand, evaluate and modify. The major requirement of a structured program is the use of single entry and single exit control structures: the most obvious difference between structured and non-structured programs is the absence of GOTO statements. Modularity breaks the complete program up into modest size sub-programs each of which performs one specific function. Each module should be self-contained and should have minimum interaction with the remainder of the system. Hence, the substitution of a new module of different design for an old one will affect only the specific function of that module, leaving the rest of the system unchanged, a condition which is of considerable assistance in program development and maintenance.

One of the major factors in producing a program which is easy to understand and modify is the use of a high level programming language. The traditional disadvantages of a high level language when compared with assembly language — requiring more memory and running more slowly — have been offset to a large extent by the development of modern optimizing compilers. The added advantages of clarity and greater productivity with high level languages now give them a clear superiority for the type of system dealt with here. The choice of a particular high level language depends on several factors:

- (i) It must possess the control structures to allow the use of structural programming.
- (ii) It must be supported on the hardware in use and allow easy access to services provided by operating system or executive.
- (iii) It must be supported by a strong international language standard to ensure portability of software to different hardware should this become desirable at some time in the future.

The only high level language which meets all of the above requirements is FORTRAN 77 (ANS Fortran X3.9—1978). This language has the added advantage of being familiar to most of the people involved in the original development of the system. It was therefore decided to use FORTRAN 77 for all programming of the data acquisition system, as has been done for most similar installations around the world (see for example References 2 and 3).

3. HARDWARE

The data acquisition system hardware is based upon a Digital Equipment Corporation PDP-11/44 central processor. The PDP-11/44 is a fourth generation 16-bit general purpose

computer and includes 8 Kbytes of high speed cache memory to provide fast program execution and high system throughput. In its current configuration the central processor is equipped with 512 Kbytes of error correcting memory and includes a floating point processor. The central processor provides interface parts for a console terminal and a TU58 cartridge tape drive used mainly for diagnostic purposes. At the present time the peripheral configuration is (see Figure 1):

- (i) Dual RLO2 removable disc drives each with a formatted capacity of 10 Mbytes. Presently one of these drives is used as the system device, containing all system and executive routines as well as space for program development. The second drive is used as the primary storage device for data.
- (ii) Dual RXO2 floppy disc drives each with a formatted capacity of 512 Kbytes. These drives are used to archive both data and source programs.
- (iii) A Facit 4542 250-character per second dot matrix line printer.
- (iv) An 8-port serial line multiplexer providing access to the system for up to eight devices using RS232C standard protocol, operating at speeds of up to 19.2 Kbytes per second. At present three of these serial ports are used for general input/output using visual display terminals. One port is used to support a colour graphics terminal with a resolution of 1024×780 and another to support an HP7221 multi pen plotter. Finally, one port is connected to the output of the existing data serializer installed in 1970.
- (v) Two independent DR11-C 16-bit wide parallel input/output interfaces. These parallel interfaces will be connected to a locally developed data bus which will provide read/write access to all wind tunnel instrumentation.

Experience with the system since its installation in 1982 has indicated that although the present configuration is generally satisfactory, the utility of the system (and tunnel productivity) would be greatly improved by the addition of two major items.

- (i) A large capacity, fast, fixed disc to be used as the system device. Experience has shown that the use of an RLO2 disc as the system device is not entirely satisfactory due largely to its limited capacity. A new drive of approximately 150 Mbytes capacity would allow a much more user-friendly system to be provided. Such an addition would also allow both RLO2 drives to be used for data storage making possible concurrent acquisition and post-processing of data from different tests, and increasing the system's ability for on-line comparison with data from other sources.
- (ii) An industry standard magnetic tape drive. As the system has developed and tunnel productivity increased, the amount of data generated has exceeded that expected when the system was initially specified. This has meant that the floppy disc drives with their limited capacity, have reached the limit of usefulness as archival devices. An industry standard magnetic tape would also increase the ability to transfer data in machine readable form to customers.

Interaction between the software and the particular hardware configuration has generally been negligible. The use of a high-level programming language together with the comprehensive services provided by the operating system and executive (see next section) has allowed a system to be developed which is very nearly machine independent. The one area in which the hardware has had a considerable influence on system design is that of the form of input of data from the wind tunnel instrumentation. The data acquisition system which existed when the new system was installed in 1982 collected data from manually selected instrumentation sources and output it in a prearranged sequence and in serial form. (Hence this device is known as the data serializer). The initialization of a data output sequence was under the control of the operator via an electrical push-button. Hence data become available at the data serializer output in an asynchronous

manner: the device looking at this output has no control over the timing of the data stream and only becomes aware of the beginning of a data system by its appearance at the output. To enable the PDP-11/44 to be used in the data acquisition system as quickly as possible, without a lengthy tunnel shutdown, it was decided to use the existing data serializer as the source of instrument data as a short-term solution. As mentioned above, two parallel input/output interfaces were provided as part of the system peripherals. Ultimately it is intended to use these parallel interfaces together with a locally developed digital data bus as the primary input/output interface between the central processor and all tunnel instrumentation. The data bus provides full 16-bit wide input and output plus control and addressing functions. It has been designed to allow input or output to be sent to or received from any individual tunnel instrumentation source, in any order, and with the timing controlled by the central processor software. When fully implemented, this bus should greatly increase the flexibility of the data acquisition.

To avoid the need to shut the tunnel down for long periods while a changeover is made from one input/output source to another, the development of the system has been planned in three stages:

- (i) All data are received from the data serializer connected to the central processor via an RS232C serial line port.
- (ii) Data are received from the digital data bus and the data serializer, with instruments being transferred from the serializer to the digital data bus as and when hardware becomes available.
- (iii) The final arrangement with all data being available on the digital data bus.

At the time of writing, the system has just entered stage (ii) with the first inputs becoming available on the digital data bus.

Besides an increase in flexibility, the full implementation of the digital data bus will also provide a worthwhile decrease in the time taken to acquire data. However, the ability to produce output to an instrument and to control the timing of data transfers by means of the central processor, will be far more important. Both of these facilities will allow a further concentration of user communication with the system into a single device (the operator's visual display terminal), greatly increasing the user-friendliness of the overall system.

4. SERVICES PROVIDED BY THE OPERATING SYSTEM AND EXECUTIVE

The operating system chosen for supporting the data acquisition system is Digital Equipment Corporation's RSX-11M (Refs. 4, 5). RSX-11M is a multiuser multitasking operating system optimised for efficient real-time programming. Multitasking, the concurrent processing of two or more tasks residing in memory, is accomplished via a priority ordered queue of tasks demanding system resources. While it is true that only one task can have control of the central processor at any instant, concurrent execution of several tasks can be achieved since other system resources, particularly input/output device operations, can execute in parallel. While one task is waiting for an input/output operation to complete, for example, another task can have control of the central processor. Task execution is under the control of the operating system and is event driven. A task retains control of the central processor until interrupted by a task with higher priority, or until it becomes unable to continue (e.g. waiting for an input/output operation to complete). Task priorities may be set in the range 1 to 250, with high priority real-time tasks close to the upper limit. Additionally, for tasks of equal priority, an equitable share of central processor time is allocated by the round-robin scheduler. At regular intervals (every 1/50th second) the scheduler checks that no other task of similar priority is waiting. Operating tasks have their priority reduced at every scheduler interval (over a limited range) and thus no single task is able to dominate the central processor to the exclusion of all others. This round-robin process is only applied to tasks with priorities of 150 or less, and hence real-time tasks which must retain use of the central processor until completion are not affected.

As well as controlling processor time allocation, the executive also provides the primary interfaces between the hardware and a program running on the system, and between the hardware and the people who use the system. Among these services are such functions as memory allocation, device drivers, data and file management, system utilities and programmed system services. The RSX-11M operating system provides a comprehensive set of programmed system services called executive directives. These executive directives allow FORTRAN programs access to many of the services provided by the executive and play a very important role in real-time programming. The executive directives of immediate use in the design of the data acquisition system will be described in some detail.

- (i) Task execution control. This group of directives allows one task to control the execution of other tasks. A task may start another task (REQUES), request that another task be run repetitively at predetermined intervals (RUN), or stop the execution of another task (STOP, ABORT).
- (ii) Task Status Control. The most useful directive in this group gives a task the ability to dynamically change the priority of another task or of itself (ALTPRI).
- (iii) Event associated directives. This group of directives provides inter- and intra-task synchronization and signalling. This is achieved through the use of event flags which are shared by some or all tasks which are currently awaiting execution. Event flags may be set or cleared (SETEF, CLREF) which return as well the previous state of the event flags. A task may wait for one (WAITFR) or more (WFLOR) even flags to be set by the executive or by another task, thus allowing synchronization between tasks. A task may also suspend its own operation for a finite period before once again competing for system resources (WAIT or MARK).
- (iv) Intertask Communication. The inter-task communications-related directives allow a task to send and receive message packets to or from another executing task (SEND and RECEIV). Data is transmitted in the form of 13-word FORTRAN integer arrays. In order to synchronize the transmission of messages between two tasks, the sending task specifies an event flag in the send data directive, the event flag being set when the executive is ready to pass the message packet to the receiving task. The receiving task specifies the same event flag in a wait for event flag directive and may then receive the message packet once the event flag is set.
- (v) Input/Output Communications. These directives (QIO and QIOW) allow tasks to access input/output devices at the driver interface levels, thus allowing direct interaction with the device driver for operations which are not normally available in the FORTRAN operating environment.
- (vi) Parent Offspring Tasking. These directives allow tasks to start other tasks, passing commands to them if required, and to receive status information. For present purposes the most important of these directives is the spawn directive (SPAWN) which requests activation of another task. This directive has additional functions which are not provided by the RUN or REQUES directives described above: a spawned offspring task may be a command line interpreter (CLI) and the spawned offspring task can return current or exit status information to its parent task.

The above is far from a complete list of directives provided by the executive, excluding several important groups (notably memory management and trap-associated directives). A complete description as well as details of the use of all directives may be found in Reference 4.

The use of executive directives provided by the particular operating system in use obviously makes the data acquisition system software machine dependent. However, care has been taken to use only those functions which could reasonably be expected to be available, either in directly equivalent or compatible forms, on any machine and operating system designed for use in a

real-time environment. Since the use of executive directives is by CALL's to system provided FORTRAN subroutines, conversion to a different operating system would be relatively straight forward.

Before completing the discussion of services provided by the executive, the topic of system traps must be considered. System traps are transfers of control (sometimes called software interrupts) that provide tasks with a means of monitoring and reacting to events. There are two kinds of system traps:

- (i) Synchronous System Traps (SSTs).—SSTs detect events directly associated with the execution of a program. They are synchronous because they always recur at the same point in the program when trap-causing instructions occur. Illegal instructions or instructions with invalid addresses cause SSTs, and most SSTs are associated with some sort of error condition. The operating system takes care of SSTs with no intervention from the user, and no further consideration need be given to them here.
- (ii) Asynchronous System Traps (ASTs).—ASTs detect events that occur asynchronously with a task's execution. That is, the task has no direct control over the precise time that the event — and hence the AST — may occur. The primary purpose of an AST is to inform the task that a certain event has occurred — for example, the completion of an input/output operation. ASTs are dealt with by means of short, user written (usually machine language) subroutines to which control is transferred when the AST occurs.

The only use of ASTs in the system as it has been developed so far, is in monitoring terminals for unsolicited input. Without the use of an AST, a task must continually monitor the input from a terminal if it is not to miss any unexpected user input. This is not only wasteful of system resources, but can also slow up the operation of the system to a considerable extent. To avoid this situation, an unsolicited input AST may be provided. When any character is typed on a terminal, an interrupt occurs and control is transferred to the AST. The AST sets an event flag and any subsequent characters are stored in a buffer. Meanwhile, the main task can continue with its execution, checking the event flag occasionally. When the event flag is set, the task can retrieve the typed characters from the buffer, carry out any processing specified by them, and then return to place where it was interrupted and continue. This process may be extended to allow one task to monitor the input from more than one terminal without needing to continuously monitor any of them.

5. DATA STRUCTURES

Modern wind-tunnel data acquisition systems are capable of producing a very large quantity of data. It is not unusual to acquire twenty polars per day, with each polar containing twenty-five model attitudes. Since each model attitude considered may include up to twenty-five individual pieces of data, it is quite possible for a single day's running to produce 10 000 or more individual data items. With such a rate of data generation, it is obvious that considerable thought must be given to the design of an efficient data structure before proceeding with software design. This section describes the data structure which has been implemented in the low-speed tunnel data acquisition system.

In general, a wind-tunnel test program will involve testing over many independent variables. Independent variables of interest could include model configuration, model attitudes, control surface settings and Mach or Reynolds number. In a given test, particular combinations of independent variable values are set up in the wind-tunnel, and one or more (usually more) dependent variables measured, processed and recorded. Recent experience has indicated that for a comprehensive full-model aircraft test program, up to seven independent and twenty-five dependent variables are necessary. The most common approach to obtaining all the required

combinations of independent variables is to vary only one at a time with all others held constant, then to repeat this variation with some different set of constant values. The data structure has therefore been designed to be hierarchical, successive levels being characterized by one or more independent variables, with the frequency of variation of the independent variables decreasing with each level. Thus the highest level is characterized by the independent variable varying least frequently -- in most cases the model configuration -- with successively lower levels being characterized by independent variables varying more and more frequently.

In practice, the data structure makes use of groups of files at the highest level, each group of files containing data for a particular model configuration. At the next level, the data included in each file relates to a particular combination of values of independent variables known as "file constants". Experience has shown that at this level there is often little significant difference between the frequencies of variation of several independent variables, and this level may therefore be characterized by up to four "file constants". Within each data file, individual data records are grouped into "blocks", each record in a data block being associated with the same value of an independent variable known as the "data block constant". At the lowest level, records within a data block contain data for successive values of the most frequently varied independent variable known as the "data block variable".

Figure 2 presents an example of the implementation of the data structure for a test in which the independent variables are model configuration, Mach number (M), control surface deflection (δ), angle of sideslip (β) and angle of incidence (α). It is assumed that for this test programme, each wind tunnel run consists of measurements at a range of values of angle of incidence, with this range being repeated for a range of values of angle of sideslip. Hence, for this test, the data block variable is angle of incidence and the data block constant is angle of sideslip. Each data file thus contains blocks of data at constant values of angle of sideslip, with records within each block recorded for a particular value of angle of incidence. Each file is characterized by two file constants, Mach number and control surface deflection, inferring that the model attitude variation would be repeated for each of the required combinations of these two variables. The group of files shown in Figure 2 are all associated with a single model configuration, with a further group of files associated with every other configuration considered.

Each data file contains a file header block consisting of three data records which completely describe the contents of the file. This information includes the number, names and values of the file constants, the names of the data block variable and data block constant, and a complete description of the contents of each data record. This is in the form of the number of variables in each record (which must be the same for each record in the file) and a name to be associated with each of these variables. The intention of the file header is to allow programs which use the data in these files, such as plotting programs, to interact with their users in a most straight forward manner. The user may refer to any piece of data in the file directly by name and does not need to have a precise knowledge of the detailed contents of the file.

Experience with this data structure has shown it to be capable of accommodating all test programs so far encountered in the low-speed wind-tunnel. Its generality should allow its application to data from other wind-tunnels with little or no modification. In addition, since its implementation, a considerable improvement in the quality of test program design has been noted, presumably brought about by the need to fit the test program into the logical framework of the data structure.

6. SOFTWARE

This section describes details of the data acquisition software. It is not intended to be a user's guide or to provide descriptions of algorithms and computational procedures: these are maintained in machine-readable form on the central processor to allow simple revision to reflect the most recent version of the software. The intention is rather to give the reader an overall impression of the function of each component of the system and how they interact.

A data acquisition system must include, at the very least, the following basic functions:

- (i) It must gather the raw data from the wind-tunnel instrumentation. In the present situation, the data may be available on either the data serializer or on the digital data bus. The system must be able to determine from which source a particular piece of data is available, and accept it at the desired instant.
- (ii) It must be able to process the raw data in accordance with the requirements of the particular test.
- (iii) It must produce data files suitable for input to programs which will be used for post-test data analysis, for example to produce plots and listings of the data for delivery to customers and for publication in test reports. In addition, output data should be in a form which allows for efficient long-term storage and archiving.

As well as these basic functions, it is desirable that the design of the system should include the following:

- (i) The ability to produce real-time display of the data as it is being recorded, as well as an interactive interface with the user to allow the tailoring of the display to the immediate needs of a particular test. Such real-time display of data allows the user to detect data which have been subject to any instrumentation malfunction and to decide to repeat the point while the test is in progress. The ability to delete such points from the data is also highly desirable.
- (ii) The ability to recompute the data obtained in a particular run at some later time. Such off-line recomputation is useful to avoid having to discard otherwise satisfactory data due to an error in processing or parameter specification.
- (iii) The minimization of the number of discrete software packages. The number of different types of test which can be carried out in a wind tunnel is quite large. If a separate software package must be used for each different type of test, a decrease in tunnel efficiency will inevitably result, due both to lack of operator familiarity with the software, and to the increased probability of little used packages containing undetected errors. Hence each package must be as general as possible and as much common code as possible used for every type of test.

The PDP-11 44 is basically a 16-bit machine. Although it may address up to 4 Mbytes of memory, the address space available to any single task is limited to 64 Kbytes (65536 (64 K) being the largest number which can be represented by 16 bits). At a very early stage in the development of the software, it was realized that the size of the program plus data storage areas would vastly exceed this maximum. There are several methods available to overcome this limitation (dynamic memory mapping, disc or memory overlaying etc.) but it was decided to take advantage of the intended use of the operating system as a multi-tasking one. Hence the system software has been written as a series of interlinked but separate tasks. Since each task may use up to 64 Kbytes of memory, the total memory need by the software is only limited by the number of separate tasks in use (and to some extent by the available total memory capacity if time consuming disc to memory transfers are to be avoided). This approach also makes use of the advantages of parallel processing mentioned previously.

The system software in use for any particular wind tunnel test therefore consists of two groups of tasks:

- (i) A common group of tasks which is always present regardless of the type of test. These tasks control all input of data from tunnel instrumentation and store them in a raw data file. They provide the primary user interface with the system and control the interaction and synchronization of all other tasks.

- (ii) A group of tasks which is job-specific, i.e. they change with the type of test. This group of tasks carries out all job-specific computation and display functions, and produces all job-specific (computed) data files. At present three job-specific groups of tasks have been written: one to accommodate all tests in which forces are measured be it by strain-gauge balance or underfloor balance (FRC), one to accommodate pressure measurement testing of two-dimensional aerofoils (2DP) and a third to accommodate pressure testing of three-dimensional objects (3DP). To allow these job-specific tasks to be as flexible as possible, all are designed to obtain details of the particular test from "configuration files". Configuration files contain information on what data are to be measured and from where they are to be obtained as well as calibration data for strain gauge balances, transducers etc.

With the total software divided up into many individual tasks, procedures must be devised for intertask communication. At present, intertask communication is conducted at three distinct levels:

- (i) Inter-task communication through the setting and reading of event flags. This is the lowest level of communication and is used mainly for intertask synchronization. One task sets an event flag to indicate to another task that it has completed some operation. The second task can then wait for the event flag to be set before proceeding with an operation which requires that the first task has completed its operation. Event flags are also used to indicate that some exceptional event, such as an operator or instrumentation error, has occurred.
- (ii) Parameter transfer through the sending and receiving of message packets. This mode of intertask communication is used by a controlling task to inform some sub-task of the type of operation required of it. The controlling task first fills a thirteen-word message array with the parameters to be sent. It then invokes the SEND directive, specifying event flag 'n'. When the controlling task can no longer proceed without completion of the sub-task's function, it waits for the sub-task to indicate its readiness by setting another event flag 'm'. When the sub-task is ready to receive a message, it waits for the controlling task to set event flag 'n' and when set invokes the RECEIV directive to dequeue the next message. It interrupts the parameters in the message array, carries out the required action, and when finished sets event flag 'm' to indicate that the controlling task may proceed. The sub-task then returns to wait for the arrival of another message. With this type of communication, quite complex operations can be carried out, with significant overlap in controlling and sub-task execution.
- (iii) Large-scale data sharing through the use of a global common area. This highest level is used for most intertask communication of data arrays. The global common area is not unlike FORTRAN common, but instead of individual sub-routines sharing the data it contains, the data may be shared by many individual tasks. The global common area is an 8 Kbyte area of memory accessible to all tasks which have specified the resident common option at task build time (see Reference 5, Chapter 5 for details). Tasks may have read only or read/write access to the global common, and hence it may be protected from corruption by tasks which need only have read access. By linking to the global common area each task's address space is of course reduced by 8 Kbytes (to 56 Kbytes) but the ability to share data storage with any other task far outweighs any disadvantage.

The remainder of this section will indicate how the requirements and concepts discussed so far have been realized. To do this, the operation of some of the tasks written for the system will be presented in some detail.

Consider first the common (non job-specific) group of tasks. The main task associated with this group is called DATAIN, and as far as the operator is concerned, this is the only task running.

When the operator runs DATAIN, its operation is as follows (see Figure 3). DATAIN first enquires of the operator the type of job he wishes to run: at present the options are FRC for a force measuring test, or 2DP or 3DP for two- or three-dimensional pressure measuring tests. All other common group subtasks and job-specific sub-tasks required for the particular type of job are then activated and initialized through SEND/RECEIV directives. The mode of operation must then be input. DATAIN has two modes of operation: on-line and off-line. For on-line operation, the tunnel must be running and data is acquired directly from the tunnel instrumentation. In the off-line mode, system operation is almost identical, except that the tunnel need not be running, and all data are obtained from a previously recorded raw data file.

When operating on-line, DATAIN informs the operator of its readiness with a message on the operators terminal. The operator has two options: he can press the data serializer "record" push button to record a block of data, or he may type a command at his terminal. The commands available to the operator are:

- (i) Display sub-task parameter modification (*D*). This command allows the operator to change the appearance of the real-time data display through changes to scales, axes, quantities displayed etc.
- (ii) Terminal display modification (*T*). This command allows the operator to modify the format and contents of the data presented on the operators terminal.
- (iii) Plot the present display (*P*). This command will produce a hard-copy of the current display.
- (iv) Block end within a tunnel run (*B*). This command is used to indicate the end of a data block within a multi-block tunnel run.
- (v) Continue data file with next tunnel run (*C*). This command is used to indicate the end of the current tunnel run. It is used when the next tunnel run will begin in a short time and data are to be stored in files with the same names as those of the run just completed. The command avoids the overhead involved in the initialization dialogue of the main task and all sub-tasks.
- (vi) Reject the last set of data collected (*R*). This allows the operator to delete from all files any recorded data in which he detects an error, or considers to be suspect in any way. Only the immediately previous data point may be rejected and the data point will be erased from the display if it is in use.
- (vii) Abort the run (*A*). This command allows the operator to immediately end a run when for example a serious malfunction is detected. All data recorded up to that time will probably be retained but whether such data are of any use will depend on the circumstances.
- (viii) End the current file (*E*). This command informs the system that the current tunnel run is completed and that no further data will be added to the data files at least for the time being. Files which have been "ended" can be added to at a later time by specifying an option to "append to an old data file" during the initialization dialogue.

Input of the single character memory corresponding to any of these commands is detected by an unsolicited input character AST routine as described in Section 4. On detecting a typed character this routine sets event flag number 73 causing DATAIN to branch to its command interpreting routine (CMDINT) where the character is retrieved from the type-ahead buffer and the required action undertaken. Activation of sub-tasks to carry out the desired action is achieved through SEND/RECEIV directives. The contents of the message packet and their meanings are given in Figure 5. (Note that the same format is also used for actions other than

those directly accessible to the user by typing a command). If the command was Abort or End, DATAIN awaits for all sub-tasks to complete the processing and then exits itself. Otherwise, once the sub-tasks have indicated completion DATAIN informs the operator of its readiness once again and the process is repeated.

If, instead of typing a command, the user presses the record pushbutton, the process takes a quite different course. Another member of the common group of tasks BUFFIN, which is requested by DATAIN at initialization, detects the beginning of output from the data serializer with another unsolicited input character AST routine. When this routine sets its event flag, BUFFIN reads a line of data from the serializer and places it in a seven-line circular buffer in the global common area. For each line placed into the buffer, BUFFIN sets a unique event flag (with numbers in the range 65 to 71). The setting of one of those event flags indicates to the serializer data decoding routine (INPUT) in DATAIN that that line may be removed from the buffer, decoded and placed in the correct position in global common for other sub-tasks to make use of in their processing. The rate at which INPUT can decode serializer data has been found to be sufficient to avoid BUFFIN overwriting data before they can be decoded. Note that a timeout period is also begun when the first line of data is received from the data serializer to avoid the system waiting forever in the case where for some reason the data serializer does not output the expected amount of data. If this time period expires before all the expected data are received, a message is sent to the operator's terminal and the process repeated.

Before processing the first data serializer line, INPUT instructs all sub-tasks to initiate input from the digital data bus. This input is carried out in parallel with input from the serializer and INPUT does not check for its completion until it has completed decoding the serializer input. The operation of the digital input/output task DIGIO, also part of the common task group, is described in Appendix A. Also at this time, raw data gathered from both the data serializer and the digital bus on the previous record cycle are output to the raw data file. The delay in output of raw data is necessary to allow the operator to Reject the previous data line if he so desires.

Once all data are input and decoded, they are checked for validity and then computation and display job-specific sub-tasks are instructed to carry out the required processing, once again via SEND/RECEIV directives. When the sub-tasks indicate completion, DATAIN returns to await the input of another command or the beginning of another data input cycle.

When operating off-line DATAIN reads data from previously recorded raw data files and processes them exactly as described above. However in this mode, no operator input commands are available, and some important ones must be simulated. The End command is easily simulated by detecting the end of file on the raw data file. Conditions representing the Continue or Block commands may only be detected by the job-specific sub-tasks. Hence before reading the next group of lines from a raw data file, DATAIN checks the state of event flag 95 which will have been set by a job-specific sub-task if one of these conditions existed for the data last processed. In this case DATAIN instructs the sub-tasks to take the appropriate action, once again using SEND/RECEIV directives.

DATAIN thus provides all the capabilities required of a data acquisition system. As many as possible of the functions common to all wind tunnel testing have been compacted into a single common group of tasks. Although there may be as many as six individual tasks active during a test, this is hidden from the operator who appears to communicate with a single monolithic system. Only those functions unique to a particular type of test are included in the job-specific sub-tasks, and the operator's knowledge of this organization is limited to choosing the correct "job type" option for the type of test under consideration.

It is unnecessary to describe the operation of all job-specific sub-tasks in detail, since their general mode of operation is so similar that the description of one group will allow the others to be understood with little difficulty. The sub-tasks chosen for description are FRC, the group used for force measurements. This group has had by far the greatest amount of development and use, and is the most general of all the current sub-tasks.

The FRC group consists of three distinct sub-tasks: COMFRC which is responsible for the reduction of force data to a form suitable for presentation to customers or publication, DSPFRC which is responsible for the real-time display of these data and BALMON which monitors the state of loading of force balances, alerting the operator to dangerous conditions due to approaching maximum load conditions. Both COMFRC and DSPFRC react to

SEND/RECEIV directives sent by DATAIN in the form presented in Figure 4. BALMON on the other hand is simply requested by COMFRC and executes at set intervals until the tunnel run is completed.

The structure of COMFRC is presented in Figure 5. COMFRC begins execution when REQUESTed during the initialization phase by DATAIN. Once running COMFRC interacts with DATAIN via SEND/RECEIV directives using event flags 81 and 91. Event flag 81 is set by DATAIN in a SEND directive. COMFRC spends most of its time waiting for event flag 81 to be set, indicating that DATAIN has sent a data packet. When it has completed processing the instruction sent by DATAIN, or it has reached a stage where other processing may continue in parallel, COMFRC sets event flag 91. Although the flow chart in Figure 5 infers that event flag 91 is not set until each operation is complete, this has been done for clarity, and the actual stage at which the event flag is set will be noted for each process. It should be noted that there is no possibility of the two tasks getting into a "race" condition since there is always synchronization at the receipt of the next data packet.

On receipt of a data packet, COMFRC decodes the first word ISEND(1) containing the function code. The actions carried out for each value of the function code are:

- (i) ISEND(1) = 0. This code is sent by DATAIN during the initialization phase at the beginning of a run. The major functions are to obtain a file specification for the configuration file, to read and verify its contents and to obtain from the operator confirmation of any data which are illegal or inconsistent. The contents of configuration files for force measurement testing are described in Appendix B. Configuration files for other types of tests differ in the detail of their contents, but provide much the same type of information. The information in the configuration file is used to initialize many sub-routines used in the reduction and output of the data (e.g. interference correction, attitude computation and output routines). Finally, various counters and flags are initialized to their starting values. The completed event flag (91) is not set until all processing is complete.
- (ii) ISEND(1) = 1. This code is sent by DATAIN on the completion of every raw data input sequence. It is the normal mode of operation of COMFRC and consists of the reduction of all data to a form where they may be presented on the operators terminal and placed in a buffer in the global common area for use by the display sub-task. The raw data used in this process have been placed into fixed positions in global common by the input routines in DATAIN. Before using the raw data, COMFRC checks it for validity and also checks that the storage buffers in global common are not about to be filled. (The buffer size currently limits the size of any data block to a maximum of 30 records). If the mode of operation is off-line, the operation of this process is somewhat different. Firstly, if the display is not in use, no data reduction is carried out since at this stage the reduced data are only used for real-time display. Secondly, it is this process that detects the end of a data block, and provides simulation of an End or Continue command by setting an event flag (number 95). The completed event flag (91) is not set until either an error condition is detected or data reduction is complete.
- (iii) ISEND(1) = 3. This code is sent whenever a Terminal display modification command is entered. The operator is interrogated as to the changes required to the data format or content on his terminal, and the completed event flag set when finished.
- (iv) ISEND(1) = 4. This code is sent by DATAIN whenever BUFFIN detects output from the data serializer. COMFRC interacts with the digital data bus via the task DIGIO to obtain necessary data from that source. As mentioned above, this process proceeds in parallel with input from the data serializer, and event flag 91 is set upon completion.

- (v) $ISEND(1) = 6$. This code is sent whenever a Block end within run command is entered at the operator's terminal. Since the processing required consists only of resetting various counters and flags, the completed event flag is set immediately on receipt of this code to allow other sub-tasks to carry out similar processing in parallel.
- (vi) $ISEND(1) = 7$. This code is sent as a result of a Continue command being entered on the operator's terminal. When using strain-gauge sensing devices to measure model forces, it is desirable to correct the raw strain gauge bridge output for any drift with time. This is achieved by recording wind-off "zero" outputs, both before a tunnel run is begun and after it has been completed. Hence before processing the data recorded for the just completed data block, COMFRC first checks that wind-off zeros have been recorded at the end of the run. If so, the before and after zeros are averaged and used to correct the data before computing all the required quantities. If these final zeros have not been recorded, a message is sent to the operator's terminal and the "failed" event flag (number 96) is set to notify DATAIN that processing has not been completed. Once all computation has been completed, the data are output to files for later printing or further processing. In this case the completed event flag is set as soon as data processing is complete, output to data files proceeding in parallel.
- (vii) $ISEND(1) = 8$. This code indicates the receipt of a Reject command at the operator's terminal. All data in the last record are deleted from output files and global common storage areas. In this case, the completed flag is set immediately on receipt of the code and the Reject processing of all sub-tasks may proceed in parallel.
- (viii) $ISEND(1) = 9$. This code is sent whenever an Abort command is typed on the operator's terminal. As much of the data as possible is saved by attempting to close all files and COMFRC then exits. However, since an Abort command is usually entered as the result of a serious system or operator malfunction, the success of this operation cannot always be guaranteed and the state of the data may not allow any useful recovery processing. To allow Abort command processing to proceed in parallel, the completed event flag is once again set immediately on receipt of this code.
- (ix) $ISEND(1) = 10$. This code is sent as a result of an End command being typed on the operator's terminal. The processing is exactly the same as that for a Continue ($ISEND(1) = 7$) with the exception that when all processing and output are complete, rather than initializing a new data block, in this case all files are closed and COMFRC exits. The completed event flag is set as soon as the validity (i.e. the presence of a final zero) of the data has been confirmed.
- (x) $ISEND(1) = \text{anything else}$. This option is provided as a default. It should never be entered, but if it is, a logical error in the software is indicated.

The above description of the operation of COMFRC is necessarily brief and somewhat incomplete. Further information may be obtained from the operator's guide held in machine readable form on the PDP-11/44, or if necessary from the program listing which includes extensive comments. However the above description should provide an overview of the logical structure of the program in enough detail to allow future modification by someone other than the designer.

The structure of DSPFRC, the real-time display sub-task of the force test group, is presented in Figure 6. It will be seen that the overall program logic is identical with that of COMFRC with only specific details of the processing carried out for each value of $ISEND(1)$ being different. This is true of most sub-tasks for other job-specific groups (e.g. 2DP, 3DP) and in fact such similarity has been one of the major aims in designing the system. Once again, although the flowchart

in Figure 6 infers that the "completed" event flag (92) is not set until each operation is complete, this has been done for clarity, and in all processes not involving a dialogue with the operator's terminal, this event flag is set immediately the command has been decoded, and all actual display operations proceed in parallel with other operations. For cases where dialogue with the operator does take place, the event flag is set on completion of the dialogue.

Similar to the operation of COMFRC, on the receipt of a data packet, DSPFRC decodes the first word, ISEND(1), and carries out the actions indicated by this function word as follows:

- (i) $ISEND(1) = 0$. This function code is sent by DATAIN during the initialization phase at the beginning of a run. The operator is asked to provide information specifying the type of display — axes selection, data selection etc. — that he requires. He is also asked whether he requires "historical" data to be displayed and if so the specification of the file containing these data. The ability to display historical data for comparison with those currently being acquired is probably the greatest improvement over previous on-line display systems. It allows the trends of current results to be compared immediately with those of previous similar tests, and it may be used to limit the range of independent variable if only the intersection of current and historical data is required. (It is intended that in the future this facility will be extended to allow the display to include data generated numerically for direct comparison with experimental data). Finally DSPFRC draws the required axes, and if requested, the specified historical data.
- (ii) $ISEND(1) = 1$. This code is sent by DATAIN following completion of the "normal" computations carried out by COMFRC at the end of every raw data input sequence. DSPFRC adds the newly calculated data to the display in its current format having first checked that the latest data does not necessitate re-scaling of the displayed axes.
- (iii) $ISEND(1) = 2$. This code is sent on the receipt of a "Display parameter modification" command at the operator's terminal. The operator is interrogated to determine the changes required to the display format.
- (iv) $ISEND(1) = 5$. This code is sent on the receipt of a "Plot the present display" command at the operator's terminal. The current contents of the display screen are rescaled to suit the physical size of the plotter, and an output file queued to the plotter.
- (v) $ISEND(1) = 6$. This code is sent whenever a "Block end within run" command is entered at the operator's terminal. The operator is asked for details of any historical data to be displayed, the old data erased from the display and new axes drawn.
- (vi) $ISEND(1) = 7$. This code is sent as a result of a "Continue" command being entered on the operator's terminal. The reaction of DSPFRC is identical with that of (v) above.
- (vii) $ISEND(1) = 8$. This code indicates that a "Reject" command has been entered at the operator's terminal. All data from the previous record are erased from the display.
- (viii) $ISEND(1) = 9$. This code is sent as a result of an "Abort" command being typed at the operator's terminal. DSPFRC clears the display screen and exits.
- (ix) $ISEND(1) = 10$. This code is sent whenever an "End" command is typed at the operator's terminal. As for an "Abort" command, the display screen is cleared and DSPFRC exits.

- (x) ISEND(1) = anything else. This is provided to catch all illegal values of ISEND(1).
A message is sent to the operator's terminal indicating a logical error.

The final sub-task in the force measurement group of tasks is BALMON, the force balance loading state monitor. The operation of BALMON, which is quite unlike that of either COMFRC or DSPFRC, is represented in Figure 7. BALMON begins execution when REQUESTed by DATAIN in the initialization phase. Thereafter, BALMON begins a cycle of execution at set time intervals, the interval being dependent on the type of force balance in use. During each execution cycle, BALMON reads balance outputs from the digital data bus, computes the loads acting on the balance using balance calibration data previously entered into the global common area by COMFRC, and compares these loads with maximum allowable loads for the particular balance. If the maximum loads are exceeded, BALMON sends a message to the operator's terminal, and the operator may modify or terminate the test to avoid damage to the balance.

The type of monitoring carried out by BALMON has become feasible only with the advent of the digital data bus. Previously, input from the data serializer has occurred only on command from the operator. Hence the state of loading of a force balance could be checked only at the times that data were recorded. By this time, model attitudes and tunnel dynamic pressure would all be stable, and if balance loads were greater than maximum, then this situation could have existed for some time, perhaps causing irreparable damage to the balance.

This completes the description of the force measurement group of tasks. Together with the non-job-specific group DATAIN, they satisfy all of the criteria for a data acquisition system mentioned above. Perhaps the greatest attribute of the FRC group is its generality: it can handle force testing for almost all foreseeable test configurations, whether the model is mounted on an external or internal strain gauge balance, regardless of the support system in use. Thus, a single software package may be used in approximately 90% of the testing conducted in the low-speed wind-tunnel.

The remaining job-specific task groups, because they are used in only about 10% of tunnel testing, have received considerably less development. The two-dimensional pressure measurement group of tasks (2DP) provides facilities for acquiring data via mechanical pressure scanning switches ("scanivalues") from pressure tapped two-dimensional aerofoil sections. These data are converted to pressure coefficient form, integrated to give section force and moment coefficients and the pressure distribution displayed in real-time. All data are output in a form suitable for post-test plotting or other analysis as may be required. In its present form, this group of tasks can accommodate multi-element aerofoils with up to three elements, and has been successfully used in an extensive series of tests to optimise the design of flap and aileron sections of the RAAF Basic Trainer wing.

The three-dimensional pressure testing group of tasks (3DP) also acquires pressure data via scanivalues. In this case however, the only processing is the conversion of these pressure data to pressure coefficient form. Since no single form would be suitable for the full range of possible physical configurations, no attempt is made to provide a real-time display capability, and the data are simply output in a form suitable for use in post-test analysis.

7. FUTURE DEVELOPMENT

It is inevitable that the existing system will be modified and extended both in response to changing requirements and to take advantage of developments in hardware and software. Typical of these changes would be an extension of DSPFRC to allow comparisons between current experimental results and the output of computer simulations. However, significant improvements to the traditional methods of wind tunnel testing will become possible as more data sources are transferred from the data serializer to the digital data bus.

When the data from a particular source are available on the bus, an immediate advantage will result from the ability of the software to access those data whenever necessary, rather than waiting for the operator to initiate a data transfer via the data serializer read switch. Monitoring and display functions such as those carried out by BALMON for balance outputs will then be possible for other functions, greatly increasing the flexibility of the whole system. However, the

digital data bus has been designed to allow individual data source modules to be provided with significant local intelligence, and it is this area that will produce the most significant advantages. Such local intelligence could be used either to pre-process raw input data before they are passed to the central processor, or to control some function according to commands transmitted from the central processor.

To indicate the potential for reducing the tunnel operator's work load and the resulting increase in tunnel productivity, the operation of two possible intelligent data sources, model attitude and tunnel speed, will be explored in more detail.

- (i) Model attitude: Currently each of several model attitude parameters has a separate display and its own set of manual controls. Each display, including those not being used in the current test, competes for control desk space and for the operator's attention. As each model attitude parameter is transferred to the digital data bus, local displays may be deleted and the values of just those attitude parameters of interest in the current test may be sampled several times per second and used to update displays on the operator's terminal. Initially the control function would remain manual, but as the system is developed, a feedback control system could be implemented to set each attitude parameter to values specified by the central processor in response to instructions typed at the operator's terminal. Such a system would not only make the tunnel operator's job much easier but also open up the possibility of conducting testing in new and novel ways. One such possibility is the concept of constant aerodynamic parameter testing,⁶ in which attitude is controlled so as to maintain some parameter, for example lift coefficient, constant while other parameters are varied.
- (ii) Tunnel speed: Presently, tunnel speed control is achieved by manually matching the required value of dynamic pressure with that displayed on the control desk. The availability of tunnel dynamic pressure on the digital data bus would allow the data acquisition software to check that the current value is equal to the desired value (input at the operator's terminal) before beginning to record data, thus avoiding one of the most common reasons for rejecting data points. The addition of tunnel temperature and static pressure to the bus would allow their combination with the measured dynamic pressure to calculate locally the tunnel velocity, unit Reynolds number or Mach number. That quantity of most interest in a particular test could then be displayed on the operator's terminal. Again it would be possible to develop a feedback control system to control tunnel speed. Tunnel speed could then be controlled to maintain constant dynamic pressure, velocity, or Mach or Reynolds number as desired, the required value being input by the operator. Once again, such a system would not only reduce the operator's workload, but also lead to worthwhile improvements in productivity due to the faster reaction to speed perturbations caused by attitude changes than is presently possible with manual speed control.

Individually these two intelligent sources would each contribute significantly to tunnel productivity. Together they provide the possibility of a completely new approach to wind tunnel testing — the computer controlled test. The tunnel operator would enter the desired tunnel speed or set of tunnel speeds and the attitude range to be covered. Control would then pass to the computer which would maintain the desired tunnel speed while recording data at each attitude. Control would then pass back to the operator who, on the basis of the on-line data display, could either accept the run or add more data points in critical or interesting areas. Although such operation may appear somewhat ambitious, it is a highly-desirable goal which could be achieved within the next two to five years.

8. CONCLUSIONS

The data acquisition system based on a PDP-11/44 central processor has now been installed in the low-speed wind tunnel for more than two years. During that period, several major test programmes have been conducted in the tunnel, and the data acquisition system has performed satisfactorily at all times. Depending on the type of test being conducted, tunnel productivity has been increased by a factor of between three and five over that with the previous system. The time taken for raw input data to be converted to the desired form and displayed has been reduced from an average of 30 seconds (and at times of high loading, up to three minutes) when using the central site computer, to an average of a little under two seconds with the new system. It is clear that further major gains in productivity will be available in the future through improvements to the model attitude and tunnel speed control systems. Development of intelligent data source modules connected to the digital data bus will further enhance these gains.

The choice of a multi-tasking multi-user operating system has been shown to be correct, the present operating system providing all the facilities needed and being ideally suited to the type of real-time operation required by a data acquisition system. The combination of this operating system with FORTRAN 77 has also been successful. Of the more than 10 000 lines of code (excluding comment lines) which have been produced so far, only 50 lines have had to be written in Assembly Language. The use of FORTRAN 77 has produced a system which can be quickly coded, is easy to follow, and simple to maintain and modify.

Although many changes have been made to the software over its life so far, all have been changes in detail. No changes have been necessary to the overall logical design and it would appear that the present design is sufficiently general to incorporate the requirements of the data acquisition system for many years to come.

REFERENCES

1. W. F. L. Sear, C. W. Sutton and J. F. Harvey A Versatile Data Acquisition System for a Low-Speed Wind Tunnel. A.R.L. Aerodynamics Report 155, November 1980.
2. J. M. Cambra, and G. P. Tolari Real-time computer data system for the 40- by 80-foot wind tunnel facility at AMES research centre. NASA-TN-D-7970, 1975.
3. C. H. Fox Real-time data reduction capabilities at the Langley 7- by 10-foot high-speed tunnel. NASA-TM-78801, 1980.
4. ——— RSX-11M/M-PLUS Task Builder Manual. Digital Equipment Corporation, 1981.
5. ——— RSX-11M/M-PLUS Executive Reference Manual. Digital Equipment Corporation, 1984.
6. R. L. Palko, A. D. Lohr A Constant Parameter Testing Technique with Automatic Wind Tunnel Control. AIAA Paper No. 78-784, 1979.

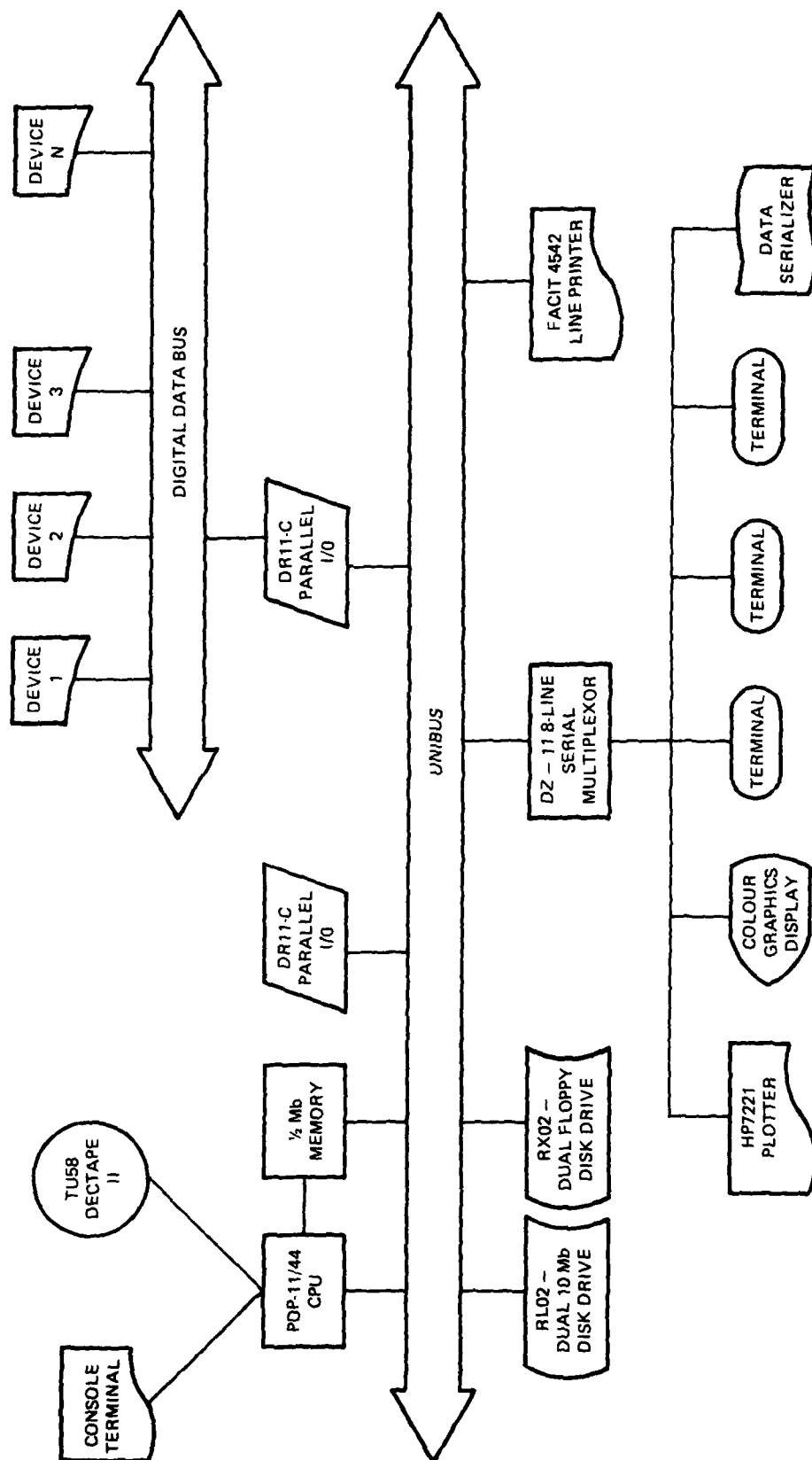
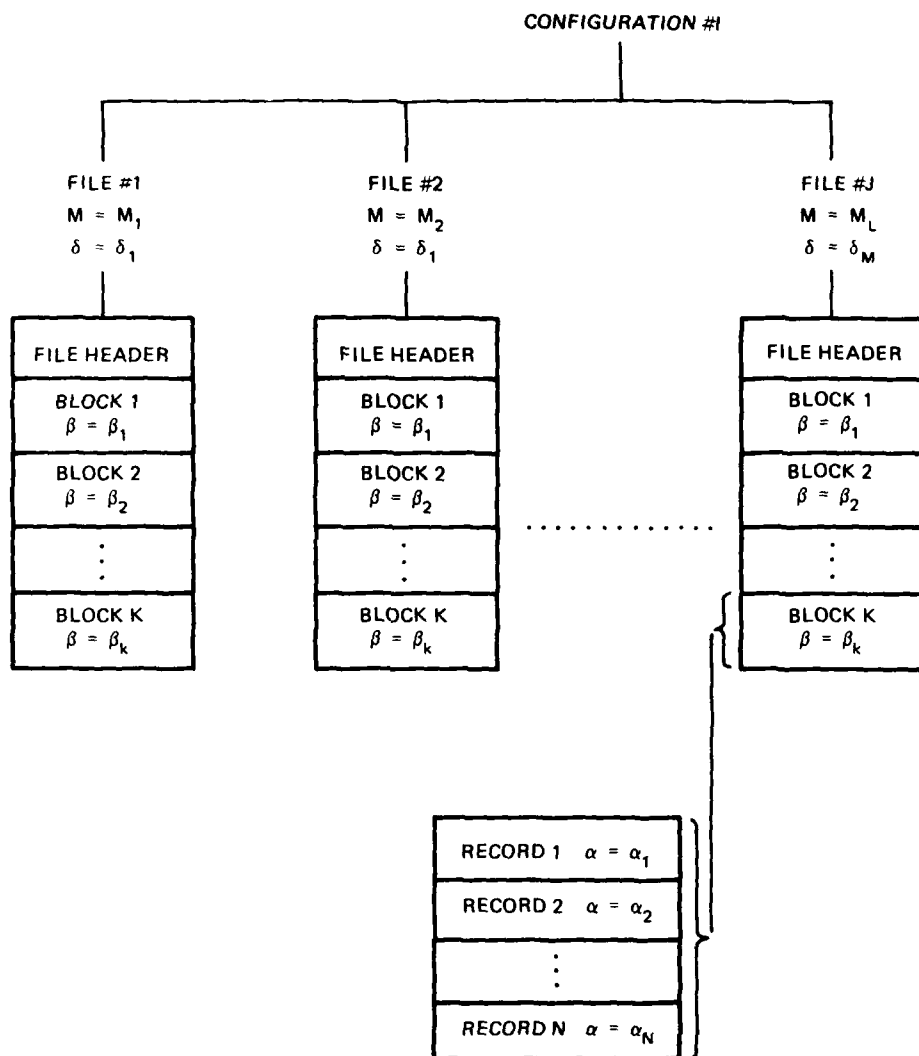


FIG. 1 HARDWARE CONFIGURATION



INDEPENDENT VARIABLES: CONFIGURATION, M , δ , β , α

FILE CONSTANTS: M , δ

DATA BLOCK CONSTANT: β

DATA BLOCK VARIABLE: α

FIG. 2 DATA FILE STRUCTURE

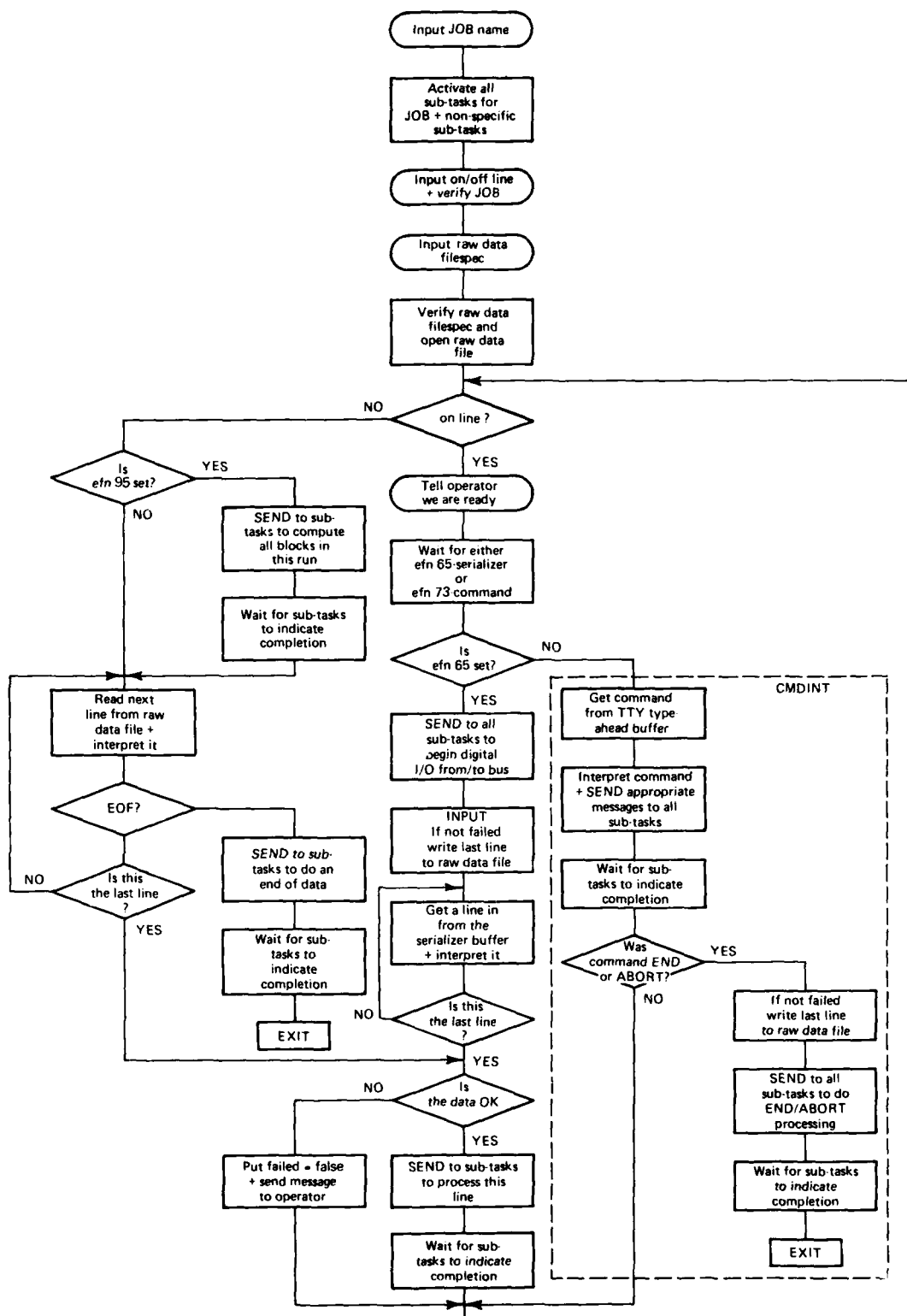


FIG. 3 STRUCTURE OF TASK DATAIN

FIGURE 4

DATAIN Message Packet Format

(a) Contents of the Message Packet

Word Number	Contents
1	Command buffer
2	= 0 if operating off-line; = 1 if operating on-line
3	= 0 if a new data file; = 1 if appending to an old data file
4-13	Unused

(b) Interpretation of Command Buffer

Contents	Meaning	Equivalent Command
0	Carry out sub-task initialization	—
1	Process a normal data record	—
2	Modify display characteristics	D
3	Modify operators terminal display	T
4	Commence digital bus data transfer	—
5	Plot the current display	P
6	Block end within a tunnel run	B
7	Continue current data file with next tunnel run	C
8	Reject the last data record	R
9	Abort this run	A
10	End this run	E

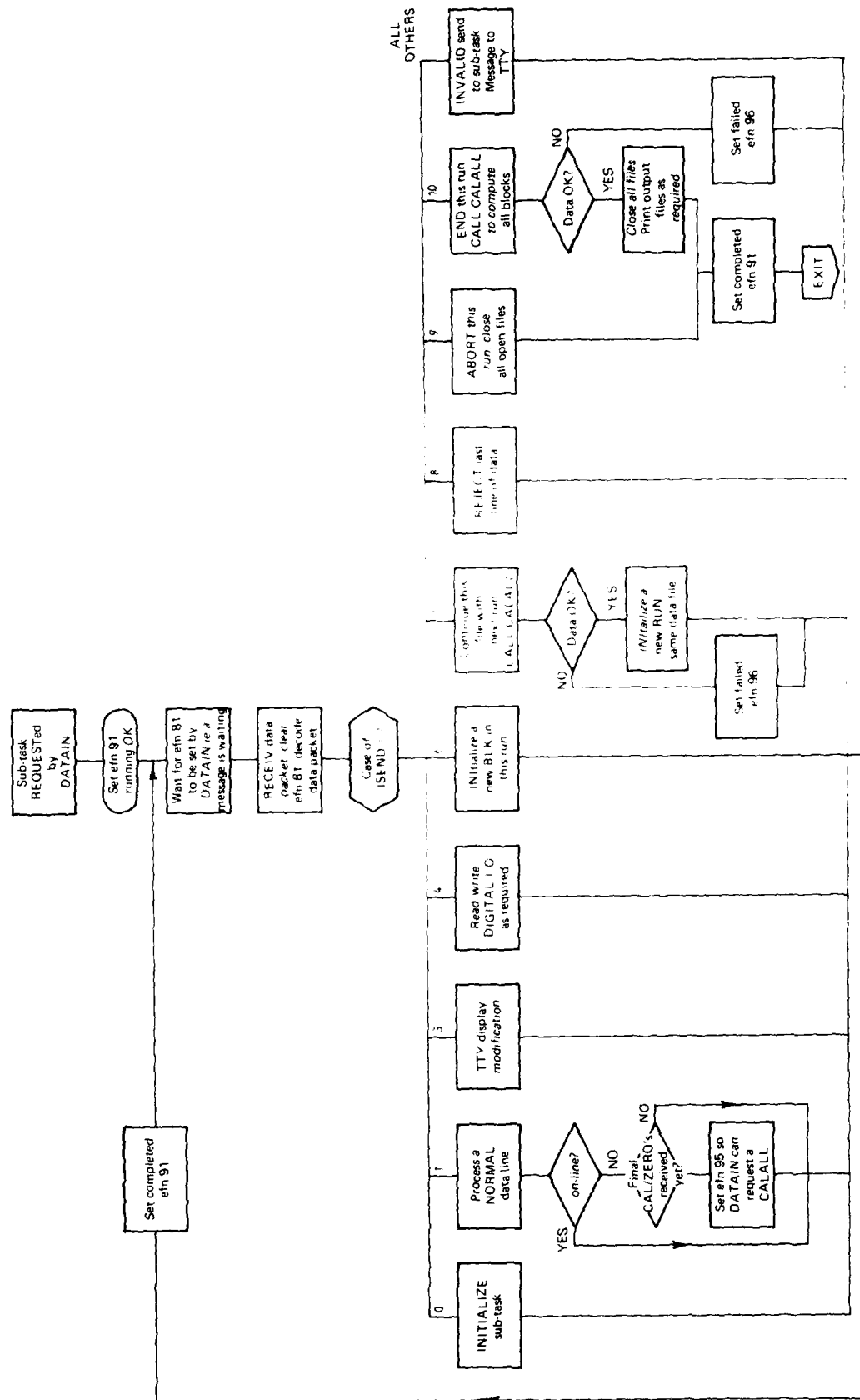


FIG 5 STRUCTURE OF TASK COMPONENT

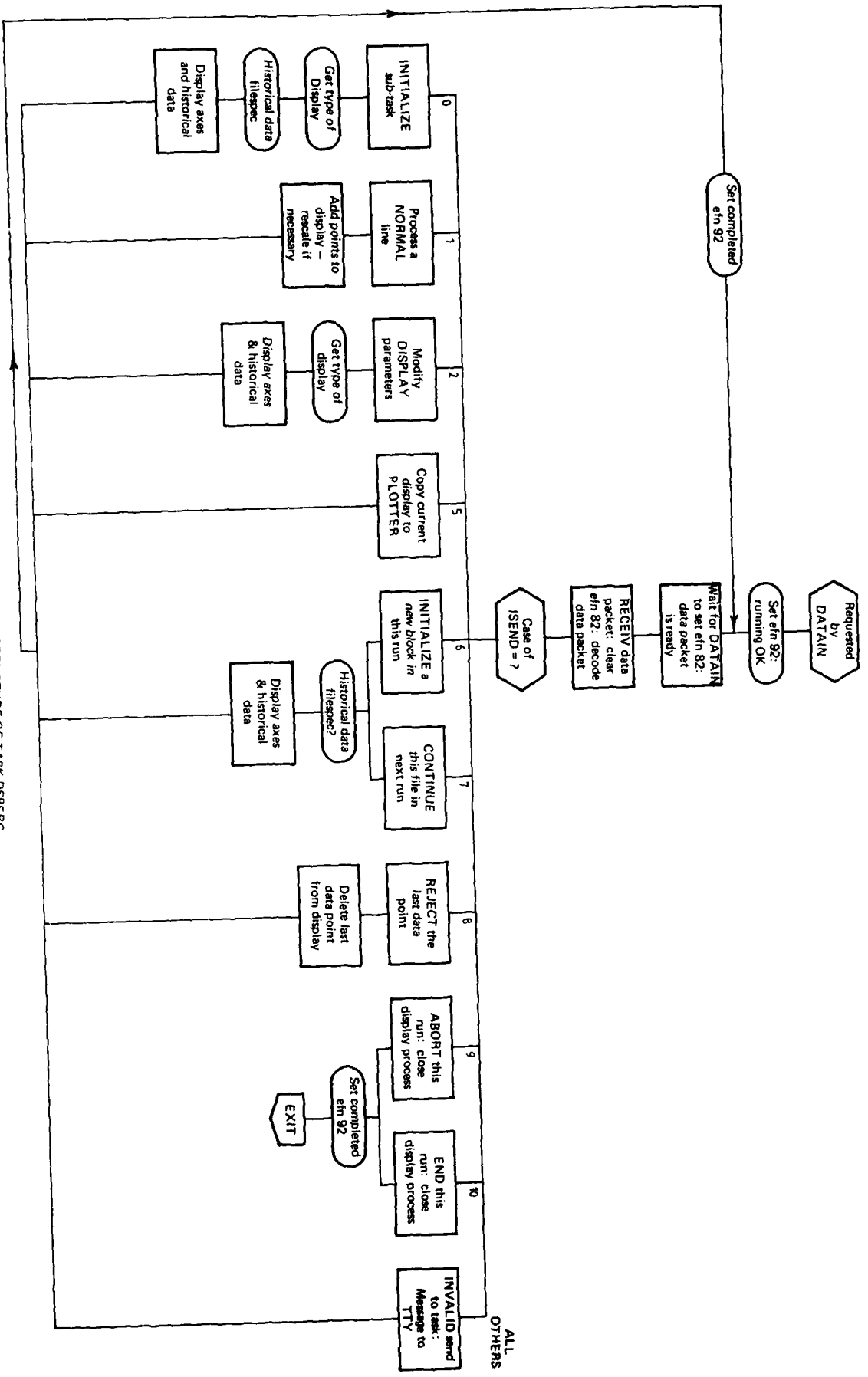


FIG. 6 STRUCTURE OF TASK DSPFRG

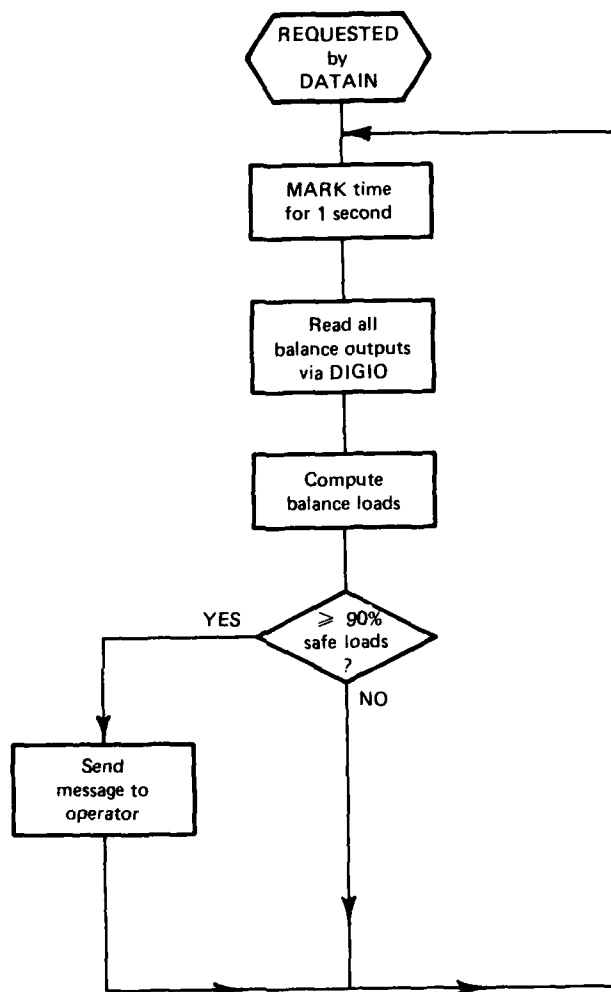


FIG. 7 STRUCTURE OF TASK BALMON

APPENDIX A

The Digital Data Bus Input/Output Task (DIGIO)

DIGIO is a task which carries out data transfers between the digital data bus and the area of memory dedicated to global common. The task is accessible to any of the common or job-specific tasks which need to communicate with the bus.

Before proceeding with a description of the design and operation of DIGIO, the arrangement and capabilities of the digital data bus itself, and its associated device driver will be described.

The digital data bus consists of 16-bit wide input and output busses and a 16-bit wide control and status register (CSR) connected to the UNIBUS of the PDP-11/44 via a DR11-C general purpose digital interface. Each module connected to the data bus is associated with one or more bus addresses, each of which may be used for either input or output. To output data to a particular address, the address bit (CSR0) is set in the CSR and the desired bus address loaded onto the output bus. Once the address is accepted (indicated by setting REQ A on the CSR) the data to be output are loaded onto the output bus (with CSR0 cleared) and the data will be transferred to the specified address. For input, the procedure is similar except that when the address is accepted (REQ A set), this also indicates that the required data are available and may be read from the input bus.

A software device driver has been written to provide FORTRAN callable subroutines to interface with the DR11-C and hence the digital data bus. These subroutines and their functions are as follows:

<i>Subroutine</i>	<i>Function</i>
KDRATT	Attaches a specified DR11-C interface to the calling task giving that task exclusive use of the DR11-C until it detaches from it.
KDRDET	Detaches the specified DR11-C from the calling task allowing other tasks access to it.
KDRSTA	Allocates a local event flag to be set whenever REQ A is set — also sets INT ENB A which causes an interrupt to be initiated whenever REQ A is set.
KDRSTB	Same function as KDRSTA but acts on REQ B.
KDRCLA	Clears INT ENB A and disassociates event flag from the interrupt.
KDRCLB	Same as KDRCLA but acts on INT ENB B.
KDRINP	Reads data from a specified DR11-C. Contents of the CSR, output bus, and input bus are input.
KDROUT	Outputs data to a specified DR11-C. Output may be to the CSR or output bus or both. Note that output to the CSR will have INT ENB bits masked out, i.e. can only output to CSR0 and CSR1.

Hardware interrupts are simulated by associating an event flag with a particular interrupt — when the interrupt occurs, the associated event flag is set. Hence the driver does not have Asynchronous System Trap routine support and is therefore compatible with standard FORTRAN. The calling task can "see" an interrupt by waiting for the particular event flag to be set (WAITFR). Note that it is the responsibility of the calling task to clear event flags following interrupts.

DIGIO makes use of these subroutines to provide a system-wide service for any task wishing to communicate with the digital data bus.

When designing DIGIO it was necessary to consider the wide range of ways in which requesting tasks would make use of the data transfers provided by DIGIO. These include:

- (i) Use by one of the common group of tasks to output constants to an intelligent local data source during the initialization phase of a tunnel run.
- (ii) Use by a task controlling the operation of same function to send desired set point values to intelligent data modules, e.g. transmitting desired values of attitude to an automatic attitude control module.
- (iii) Use by a monitoring task to read data from a module at regular intervals, either to update a display on the operator's terminal, or to check on the state of some device, e.g. BALMON monitoring the state of loading of a strain-gauge balance.
- (iv) Use by a job-specific computation task to obtain data from the required sources when requested by the operator (or at some future time by some overall test controlling task).

Of these various uses, it is clear that those gathering data (type (iv)) must gain access to DIGIO immediately. (This will be specially important while data are input from both the data serializer and the digital data bus to ensure that data from both sources are attributable to the same instant in time). In general it is possible that all the above types of tasks (with the possible exception of type (i)) could require access to DIGIO at the same time. Thus it is necessary to associate a priority with requests for DIGIO from various types of tasks. At present, the priority may be either "high" or "low", with tasks of type (iv) making high priority requests and all the others low priority ones.

Communication between DIGIO and requesting tasks is implemented via parameter transfer through SEND and RECEIV directives, as described in Section 7. The contents of the 13-word message array used in this transfer is:

Word (1)—priority of the request (low = 0, high = 1).

Word (2)—number of an event flag to be set by DIGIO if the data transfer is completed successfully.

Word (3)—number of an event flag to be set by DIGIO if the data transfer has to be aborted or produces an error condition.

Word (4)—direction of the desired data transfer (input from the bus = 0, output to the bus = 1).

Word (5)—number of data items to be transferred.

Word (6)—index of an address buffer in global common containing the address of the first device to be accessed.

Word (7)—index of a data buffer in global common to or from which the first data item is to be transferred.

Note that word (6) and word (7) are the indices of the first address and data locations, each buffer containing as many entries as specified by word (5) following these initial entries.

Having filled the message array, a requesting task then invokes the SEND directive specifying event flag number 72 for a low priority request or event flag number 73 for a high priority request.

The structure of DIGIO is presented in Figure A1. On being REQUESTed (during the initialization phase of DATABUS) DIGIO attaches to the DR11-C associated with the digital data bus (thus ensuring the exclusive use of that interface) and associates event flags with and enables the DR11-C hardware interrupts. An attempt is then made to RECEIVE a message packet. If this attempt is successful, i.e. one or more message packets have been queued for DIGIO by the executive, DIGIO checks the high priority request event flag (number 73) and sets a high priority waiting flag to true if it is set, and then clears both high and low priority request event flags (72 and 73). If the attempt to receive a message packet fails, i.e. there are none queued, DIGIO waits for either the high or low priority request event flags to be set before attempting again to receive a message packet. At this stage it is possible that the message packet just received is a low priority request while a high priority packet remains queued. DIGIO is aware of the presence of a high priority request packet in the queue since the high priority request event flag would have been set when checked, causing the high priority waiting flag to be set to true. To avoid delaying processing of high priority requests DIGIO immediately checks the contents of word (1) of the message packet. If this word is zero, indicating a low priority request, and the high priority waiting flag is true, DIGIO abandons the processing of the low priority request, sets the failed event flag (specified by word (3)) of the requesting task, and returns to receive another message packet. High priority requests will of course never be abandoned in this way.

Once a message packet has been accepted for processing, DIGIO checks the contents of word (4) of the packet to determine the direction of the desired data transfer. For transfers from the digital data bus (read), the address of the first device to be read is obtained from the address buffer in global common using the contents of word (6) of the message packet. The address is loaded into the output buffer of the DR11-C and the address bit set in the CSR. DIGIO then waits for the occurrence of a DR11-C interrupt to set an event flag to indicate that the address has been accepted as valid, and that the required data are available in the DR11-C input buffer. The input buffer is then read, the data converted from external (complementary offset binary) to internal (two's complement integer) form and placed in the data buffer of the global common area according to the contents of word (7). The above procedure is repeated until the required number (word (5)) of data transfers have been completed. The operation of a write transfer is similar except that following acceptance of a valid address, data are obtained from the global common data buffer, converted to external form and loaded into the DR11-C output buffer. When all requested data transfers are complete, DIGIO sets the requesting task's success event flag, resets the high priority waiting event flag to false, and returns to attempt to receive another message packet.

The present structure of DIGIO does not guarantee that high priority requests will always be processed before all low priority ones. It is possible that when DIGIO checks the high priority request event flag that there are already two or more high priority requests in the queue. DIGIO will ignore all low priority requests up to the first high priority one and then process it. However, when this data transfer is complete, the high priority waiting flag will be reset to false, thus nullifying DIGIO'S knowledge of further queued high priority requests. This situation can only arise when two high priority requests are queued before DIGIO can begin processing the first one. Since high priority requests only arise as a result of a computation task's need to read data from the bus, and at present this only occurs at intervals of a minimum of several seconds, the chances of the above situation arising are extremely remote. Thus at present, this shortcoming of the task can be ignored, but may need to be considered in the future if the mode of operation of the data acquisition system is modified.

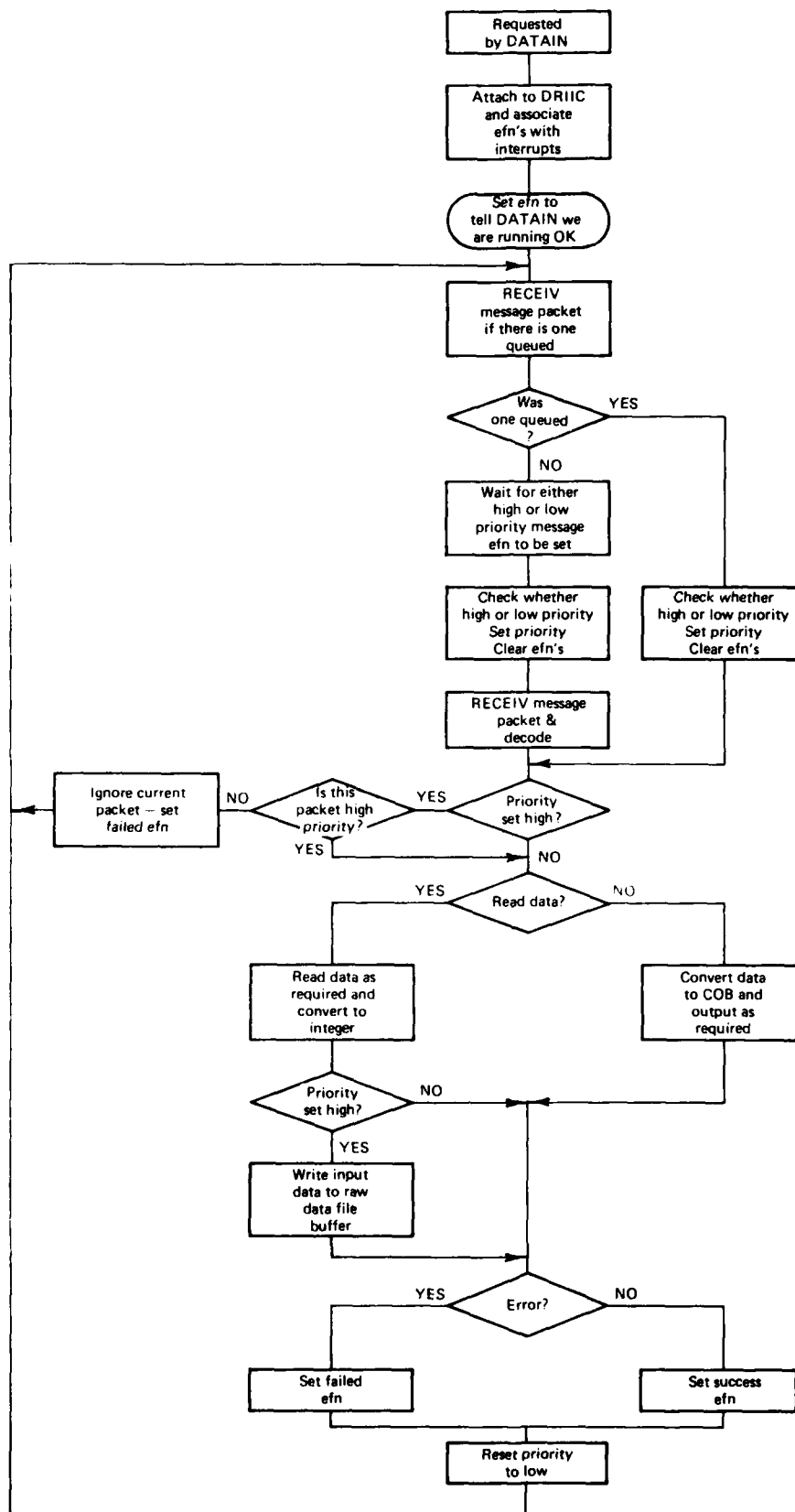


FIG. A1 STRUCTURE OF TASK DIGIO

APPENDIX B

Format of Configuration Files for Force Measuring Tasks

Force measurement task configuration files bring together all information required to acquire and compute the data for a wind tunnel test involving force measurement. These files are ASCII format to allow easy editing, and the contents of each record is:

- (i) A job title consisting of up to 50 characters used as a description of the particular test.
- (ii) A customer name containing up to 40 characters, describing the organization for whom the test is being conducted.
- (iii) This record contains information concerning the data structure:

CONFIG—a configuration reference number.

FCONST(I)—the names (12 characters) of up to four file constants.

FCVAL(I)—the values of the file constants for this file.

DBCONS—the name (12 characters) of the data block constant.

DBCTOL—the tolerance within which the data block constant is to be considered "constant".

DBVAR—the name (12 characters) of the block data variable.

- (iv)-(viii) These five records may contain up to 72 characters each of free format description of the configuration and the type of test.

- (ix) This record contains reference lengths and areas for use in non-dimensionalization:

C—the mean aerodynamic wing chord.

B—the wing span.

S—the wing area.

RENLEN—the length scale for use in determining the Reynolds number.

- (x) This record contains information required for computing blockage and lift interference:

F1, F3—blockage factors for wing and fuselage.

VOLW, VOLB—volume of wing and fuselage.

TAU1W, TAU1B, TAU2W, TAU2T—lift interference factors for the wing, fuselage and tailplane.

DELTA1—tunnel lift interference factor.

CDO—estimate of model zero lift drag.

WLCS—estimate of the wing lift curve slope.

- (xi) This record contains an integer indicating the number of movable surfaces to be considered in the test.
- (xii) This record contains information on each movable surface and is repeated for each:
- SURNAM—the name (12 characters) of the moving surface.
 - DEFSUB—the subscript (2 characters) to be appended to the surface deflection,
 - HINGE—a logical variable which if true indicates that hinge moments are measured for this surface,
 - NSG2CH—the amplifier channel number connected to the hinge moment strain-gauge bridge,
 - HFCAL, HDCAL—calibration factors for hinge moment and surface deflection,
 - REFLAN, REFA—reference length and area to be used to reduce hinge moments to coefficient form,
 - REMOTE—a logical variable which if true indicates that this surface is remotely activated,
 - NCDCH—the remote activator channel number connected to this surface.
- (xiii) This record contains information concerning any powered propeller in use:
- POWER—a logical variable which if true indicates that a powered propeller is present,
 - NAUXCH—the channel number at which propeller r.p.m. is available,
 - PROPD—the propeller diameter.
- (xiv) This record contains information on the type of model mounting and the source of attitude angles:
- MOUNT—= 1 if the model is mounted on a strain-gauge balance supported by a rear sting,
= 2 if the model is mounted on a strain-gauge balance supported via a central pylon and rear spar,
= 3 if the model is mounted on the under-floor balance,
 - NTHETA—the address at which pitch angle information is available,
 - NPSI—the address at which yaw angle information is available,
 - NPHI—the address at which roll angle information is available.
- (xv) This record contains information concerning the orientation of the balance with respect to the model.
- XBAL, YBAL, ZBAL—location of the model centre of gravity with respect to the balance moment centre,
 - THBAL, PSIBAL, PHIBAL—angular offsets of the model body axes system with respect to the balance axes system.
- (xvi) This record contains the offsets of the attitude sensors (THETA0, PSI0, PHI0) when the model body axes system is aligned with the tunnel axes system.
- (xvii) This record contains the six elements of the balance direct sensitivity matrix.

(xviii)-(xxiii) These records contain the 36 elements of the first order (linear) balance interaction matrix, $X1(I)$.

(xxiv)-(xxix) These records contain the 126 elements of the second order (non-linear) balance interaction matrix $X2(I)$.

(xxx)-(xxxv) These records contain the 36 elements of the inverse of the first order balance interaction matrix $X1INV(I)$.

(xxxv)-(xxxx) These records contain the 126 elements of second order balance interaction matrix premultiplied by the inverse of the first order balance interaction matrix, i.e. $X1INV * X2$.

(xxxxi) This record contains the five elements of the balance deflection matrix.

(xxxxii) This record contains the nine elements of the model tare weight matrix.

(xxxxiii) This record contains logical variables indicating in which form results are to be presented. Force and moment coefficients may computed in body-, stability- or wind-axes and these logical variables indicate the users choice for each component.

DISTRIBUTION

AUSTRALIA

DEPARTMENT OF DEFENCE

Central Office

Chief Defence Scientist
Deputy Chief Defence Scientist
Superintendent, Science and Program Administration
Controller, External Relations, Projects and Analytical Studies
Defence Science Adviser (UK) (Doc. Data sheet only)
Counsellor, Defence Science (USA) (Doc. Data sheet only)
Defence Science Representative (Bangkok)
Defence Central Library
Document Exchange Centre, DISB (18 copies)
Joint Intelligence Organisation
Librarian, H Block, Victoria Barracks, Melbourne
Director General - Army Development (NSO) (4 copies)
Defence Industry and Material Policy, FAS

(1 copy)

Aeronautical Research Laboratories

Director
Library
Superintendent—Aerodynamics
Divisional File—Aerodynamics
Author: B. Fairlie
M. A. Balicki
K. A. O'Dwyer
M. K. Glaister
J. F. Harvey
J. N. Hodges
N. Pollock
C. W. Sutton
J. Wattmuff

Materials Research Laboratories

Director/Library

Defence Research Centre

Library

Navy Office

Navy Scientific Adviser

Army Office

Scientific Adviser—Army

Air Force Office

Air Force Scientific Adviser
Aircraft Research and Development Unit
Scientific Flight Group
Library
Technical Division Library
Director General Aircraft Engineering—Air Force
HQ Support Command (SLENGO)
RAAF Academy, Point Cook

Government Aircraft Factories

Manager
Library

STATUTORY AND STATE AUTHORITIES AND INDUSTRY

Australian Aircraft Consortium Pty. Ltd.
Mr D. Pilkington
Mr R. D. Bullen
SEC of Vic., Herman Research Laboratory, Library
Commonwealth Aircraft Corporation, Library
Hawker de Havilland Aust. Pty. Ltd., Bankstown, Library
Rolls Royce of Australia Pty. Ltd., Mr C. G. A. Bailey

UNIVERSITIES AND COLLEGES

Adelaide	Barr Smith Library
	Professor of Mechanical Engineering
Flinders	Library
La Trobe	Library
Melbourne	Engineering Library
Monash	Hargrave Library
Newcastle	Library
Sydney	Engineering Library
NSW	Physical Sciences Library
Queensland	Library
Tasmania	Engineering Library
Western Australia	Library
RMIT	Library
	Dr P. H. Hoffman, Aero. Engineering

CANADA

NRC
Aeronautical & Mechanical Engineering Library

FRANCE

ONERA, Library

INDIA

Defence Ministry, Aero Development Establishment, Library
National Aeronautical Laboratory, Information Centre

JAPAN

Institute of Space and Astronautical Science, Library

NETHERLANDS

National Aerospace Laboratory (NLR), Library

NEW ZEALAND

RNZAF, Vice Consul (Defence Liaison)
Transport Ministry, Airworthiness Branch, Library

SWEDEN

Aeronautical Research Institute, Library

UNITED KINGDOM

CAARC, Secretary
Royal Aircraft Establishment
Bedford, Library
British Library, Lending Division
Aircraft Research Association, Library
Motor Industry Research Association, Director

Universities and Colleges

Bristol	Engineering Library
Cambridge	Library, Engineering Department Whittle Library
Manchester	Professor, Applied Mathematics
Nottingham	Science Library
Southampton	Library
Strathclyde	Library
Cranfield Institute of Technology	Library
Imperial College	Aerodynamics Library

UNITED STATES OF AMERICA

NASA Scientific and Technical Information Facility
Applied Mechanics Reviews

SPARES (20 copies)

TOTAL (120 copies)

Department of Defence

DOCUMENT CONTROL DATA

1 a. AR No. AR 003-997		1 b. Establishment No. ARL-AERO-R-163		2. Document Date February, 1985		3. Task No. DST82/022	
4. Title A REAL-TIME DATA ACQUISITION SYSTEM FOR A LOW-SPEED WIND TUNNEL				5. Security a. document Unclassified		6. No. Pages 30	
				b. title U c. abstract U U		7. No. Refs 4	
8. Author(s) B. D. Fairlie				9. Downgrading Instructions			
10. Corporate Author and Address AERONAUTICAL RESEARCH LABORATORIES, P.O. Box 4331, G.P.O., Melbourne, Vic., 3001				11. Authority (as appropriate) a. Sponsor b. Security c. Downgrading d. Approval			
12. Secondary Distribution (of this document) Approved for public release							
Overseas enquirers outside stated limitations should be referred through ASDIS, Defence Information Services Branch, Department of Defence, Campbell Park, CANBERRA, ACT, 2601.							
13. a. This document may be ANNOUNCED in catalogues and awareness services available to ... No limitations							
13. b. Citation for other purposes (i.e. casual announcement) may be (select) unrestricted (or) as for 13. a.							
14. Descriptors Wind tunnels ; Data acquisition ; Real-time operations ;						15. COSATI Group 01010	
16. Abstract A mini-computer based real-time data acquisition system designed for use in the Aeronautical Research Laboratories low-speed wind tunnel is presented. The report provides an overview of the logical arrangement of the software components of the system and describes their interaction with the mini-computer operating system, data structures, and system hardware.							

This page is to be used to record information which is required by the Establishment for its own use but which will not be added to the DISTIS data base unless specifically requested.

16. Abstract (Contd)		
17. Imprint Aeronautical Research Laboratories, Melbourne		
18. Document Series and Number Aerodynamics Report 163	19. Cost Code 546060	20. Type of Report and Period Covered —
21. Computer Programs Used		
22. Establishment File Ref(s)		

END

DATE
FILMED

1 - 86

DTIC